DE GRUYTER
OPEN

TATRA
MOUNTAINS
Mathematical Publications

DOI: 10.2478/tmmp-2014-0030
Tatra Mt. Math. Publ. **61** (2014), 105–116

USING POLY-DRAGON CRYPTOSYSTEM IN A PSEUDORANDOM NUMBER GENERATOR MSTg

Viliam Hromada — Milan Vojvoda

ABSTRACT. This paper deals with a new pseudorandom number generator MSTg proposed in 2010. Its construction is based on random covers for finite groups. We have used a public-key cryptosystem Poly-Dragon to generate these random covers and have studied the statistical properties of the resulting pseudorandom number generator by testing its output using the NIST Statistical Test Suite.

1. Introduction

Random number generators (RNGs) are one of the fundamental cryptographic primitives. The need for the generation of random data (numbers, keywords, etc.) has been constantly increasing—we need to generate more random data in less time. Basically, there are two approaches to generating random data (random bits or bytes)—there are non-deterministic random number generators and deterministic random number generators. The non-deterministic random number generators are truly "random"—in a sense that their output can not be reproduced. They usually exploit various natural sources of randomness–radioactive decay, thermal noise, the content of computer's hardware buffers, etc. Their advantage is that their output is truly random. However, they are not suitable for generating large amount of data for stream ciphers and other cryptographic primitives. The deterministic random number generators, often referred to as pseudorandom number generators (PRNGs), are not truly random. They use deterministic algorithms to generate a large output sequence of bits (*pseudorandom bit sequence*) from a certain input value (*seed*). Unfortunately, it is impossible to mathematically prove that a certain PRNG indeed produces

2010 Mathematics Subject Classification: 94A60, 68P25.
Keywords: pseudorandom number generator, public-key cryptosystem Poly-Dragon, random data, random number.
This work was supported by grants VEGA 1/0173/13, APVV-0586-11 and by A-MATH-NET.

a truly random output. That is why statistical tests have been introduced which can detect the weaknesses of such generators and hint whether such generator "behaves" similar to a non-deterministic random number generator.

In 2011, a new interesting PRNG was introduced in [3]. It is based on random covers for finite groups, which is also a relatively new concept introduced in [2]. In the article [3], authors argue that the generator is suitable for modern cryptographic applications—they carried out extensive statistical tests of the generator—we refer the reader to the article to read more details about the generator and its testing.

The generator offers great flexibility, because the generation of the random covers can be done in many ways. The authors suggest that a good PRNG should be used to generate the random covers—they used the Blum-Blum-Shub PRNG. We have chosen a different approach and have used a public-key cryptosystem Poly-Dragon to generate these random covers.

Poly-Dragon is a cryptosystem which was proposed in 2010 in [5]. It is a multivariate cryptosystem and its security is based on the problem of solving a system of multivariate quadratic equations over a finite field. This problem is also known as MQ-problem and it should be noted that it is an NP-complete problem.

As we have already mentioned, we have combined these concepts and have created a version of MSTg generator which uses Poly-Dragon to generate the random covers. We have used the NIST testing suite [4] to test the statistical properties of the resulting output sequences to see, whether this type of generator can be considered "good" in a sense that its output resembles an output of a non--deterministic random number generator.

In Section 2 we give a short description of the generator MSTg, in Section 3 we describe the cryptosystem Poly-Dragon. Section 4 deals with our construction and the results of the statistical testing, Section 5 deals with the security of this random number generator and we make concluding remarks in Section 6.

2. Pseudorandom number generator MSTg

Firstly, we start with some basic definitions which we have borrowed directly from [3] along with the notations.

Definition 1. Let $\mathcal{G}$ be a finite abstract group. The width of $\mathcal{G}$ is defined as the positive integer $w = \lceil log|\mathcal{G}| \rceil$.

Definition 2. Let $\mathcal{G}$ be a finite abstract group and $\mathcal{S}$ be a subset of $\mathcal{G}$. Suppose that $\alpha = [\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_s]$ is an ordered collection of subsets $\mathcal{A}_i \in \mathcal{G}$, such that $\sum_{i=1}^{s} |\mathcal{A}_i|$ is bounded by a polynomial in the width of $\mathcal{G}$. We call α a cover

for $\mathcal{S}$ if each element $h \in \mathcal{S}$ can be expressed in at least one way as a product of the form

$$h = g_1 \cdot g_2 \cdot \ldots \cdot g_s \tag{1}$$

for $g_i \in \mathcal{A}_i$, where $\cdot$ is the group operation defined on $\mathcal{G}$.

If the elements of the cover α (i.e., the elements of subsets $\mathcal{A}_i$) are generated randomly (i.e., chosen at random) we say that α is a *random cover*. If $\mathcal{G} = \mathcal{S}$, α is called the cover for $\mathcal{G}$.

The subsets $\mathcal{A}_i$ are called *blocks* and the vector $(r_1, r_2, \ldots, r_s)$ where $r_i = |\mathcal{A}_i|$ is called the *type* of α. The cover α is called *factorizable* if the factorization in (1) can be done in polynomial time in the width w of $\mathcal{G}$ for almost all elements of $\mathcal{G}$. Otherwise, it is called *non-factorizable*.

The assumption made in [3] that the problem of finding a factorization as in (1), with respect to a randomly generated cover, is computationally intractable. For example, let $\mathcal{G}$ be a cyclic group and $g \in \mathcal{G}$ be a generator of this group. Then one can represent the elements of subsets $\mathcal{A}_i$ as powers of the generator g. Then the factorization with respect to the cover leads to solving the Discrete Logarithm Problem in $\mathcal{G}$ [3]. If indeed this factorization is computationally intractable, then these random covers can induce functions that behave as one-way functions. This is described in [3] as follows. Let $\alpha = [\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_s]$ be a random cover of the type $(r_1, r_2, \ldots, r_s)$ for $\mathcal{G}$ with $\mathcal{A}_i = [a_{i,1}, a_{i,2}, \ldots, a_{i,r_i}]$ and let $m = \prod_{i=1}^{s} r_i$. Let $m_1 = 1$ and $m_i = \prod_{j=1}^{i-1} r_j$ for $j = 2, \ldots, s$.

Let τ denote the canonical bijection from $\mathbb{Z}_{r_1} \oplus \mathbb{Z}_{r_2} \oplus \cdots \oplus \mathbb{Z}_{r_s}$ on $\mathbb{Z}_m$:

$$\tau : \mathbb{Z}_{r_1} \oplus \mathbb{Z}_{r_2} \oplus \cdots \oplus \mathbb{Z}_{r_s} \to \mathbb{Z}_m \,,$$

$$\tau(j_1, j_2, \ldots, j_s) := \sum_{i=1}^{s} j_i m_i \,.$$

Using τ we can define surjective mapping $\breve{\alpha}$ induced by α.

$$\breve{\alpha} : \mathbb{Z}_m \to \mathcal{G},$$

$$\breve{\alpha}(x) := a_{1,j_1} \cdot a_{2,j_2} \cdot \ldots \cdot a_{s,j_s} \,,$$

where $(j_1, j_2, \ldots, j_s) = \tau^{-1}(x)$. Since τ and τ^{-1} are efficiently computable, the mapping $\breve{\alpha}(x)$ is efficiently computable.

However, given a cover α and an element $y \in \mathcal{G}$, determining an element $x = \breve{\alpha}^{-1}(y)$ is not efficiently computable if α is not *factorizable*, i.e., we are not able to determine indices $j_1, j_2, \ldots, j_s$ in polynomial time such that $y = a_{1,j_1} \cdot a_{2,j_2} \cdot \ldots \cdot a_{s,j_s}$. Once a vector $(j_1, j_2, \ldots, j_s)$ has been determined, $\breve{\alpha}^{-1}(y) = \tau(j_1, j_2, \ldots, j_s)$ can be computed efficiently. Therefore, if α is a random cover for a large subset $\mathcal{S}$ of a group $\mathcal{G}$, then finding a factorization (1) is indeed an intractable problem and the mapping

$$\breve{\alpha} : \mathbb{Z}_m \to \mathcal{S}$$

induced by α with $m = \prod_{i=1}^{s} |\mathcal{A}_i|$ is a one-way function.

Now, we will describe the generator MSTg as it is proposed in [3].

At first, we note that the structure of the group $\mathcal{G}$ is arbitrary in this generic case. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two chosen finite groups with $|\mathcal{G}_1| = n$ and $|\mathcal{G}_2| = m$ and $n \geq m$.

Let ℓ be an integer such that $\ell \geq n$. Let $k \geq 0$ be a fixed integer.

Let α be a random cover of type $(u_1, u_2, \ldots, u_t)$ for $\mathcal{G}_1$ with $\prod_{i=1}^{t} u_i = \ell$. Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be a set of random covers of type $(r_1, r_2, \ldots, r_s)$ for $\mathcal{G}_1$ with $\prod_{i=1}^{s} r_i = |\mathcal{G}_1|$. Let $\gamma = [H_1, H_2, \ldots, H_s]$ be a random cover of type $(r_1, r_2, \ldots, r_s)$ for $\mathcal{G}_2$. Assume that there are bijective mappings $f_1 \colon \mathcal{G}_1 \to \mathbb{Z}_n$ and $f_2 \colon \mathcal{G}_2 \to \mathbb{Z}_m$, which map the elements of $\mathcal{G}_1$ to numbers in $\mathbb{Z}_n$ and the elements of $\mathcal{G}_2$ to numbers in $\mathbb{Z}_m$. Then, we can define a function

$$F : \mathbb{Z}_\ell \to \mathbb{Z}_m$$

as a composition of mappings in the following way (version $\mathcal{A}$).

$$\mathbb{Z}_\ell \xrightarrow{\breve{\alpha}} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\alpha}_1} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \longrightarrow \cdots \xrightarrow{\breve{\alpha}_k} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\gamma}} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \,,$$

This mapping can be alternatively defined as follows (version $\mathcal{B}$).

$$\mathbb{Z}_\ell \xrightarrow{\breve{\alpha}} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\gamma}} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \xrightarrow{\breve{\delta}_1} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \longrightarrow \cdots \xrightarrow{\breve{\delta}_k} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \,,$$

where $\delta_1, \delta_2, \ldots, \delta_k$ is a set of random covers of type $(v_1, v_2, \ldots, v_w)$ for $\mathcal{G}_2$ with $\prod_{i=1}^{w} v_i = |\mathcal{G}_2|$.

Now we have everything in hand to describe the pseudorandom generator MSTg.

Algorithm 1: MSTg: Pseudorandom Number Generator Based on Random Covers for Finite Groups

Input : Integers ℓ, m, function $F : \mathbb{Z}_\ell \to \mathbb{Z}_m$ as defined above, a random and secret seed $s_0 \in \mathbb{Z}_\ell$, a constant $C \in \mathbb{Z}_m$

Output: t pseudorandom numbers $z_1, z_2, \ldots, z_t \in \mathbb{Z}_m$

1. For i from 1 to t do the following:
   - 1.1 $s_i = s_{i-1} + C \pmod{\ell}$
   - 1.2 $z_i = F(s_i)$
2. Return $(z_1, z_2, \ldots, z_t)$

The algorithm presented above uses a simple counter mode to generate its output—notice the constant C which is being added to s_i in each iteration. The authors of the generator suggest that any suitable mode can be used instead of the counter mode. They also argue that since F is the core function of the generator, great care must be taken when generating the covers for $\mathcal{G}_1$ and $\mathcal{G}_2$ upon which the function F is based. In other words, a good pseudorandom number generator is needed to create the random covers involved in F.

The authors in the original paper [3] used Blum-Blum-Shub generator to generate the covers. We have chosen the public-key cryptosystem Poly-Dragon instead.

3. Public-key cryptosystem Poly-Dragon

In this section we will briefly describe the Poly-Dragon cryptosystem. Poly--Dragon is a multivariate public-key cryptosystem whose security is based on the problem of solving a system of multivariate quadratic equations over a finite field (MQ-problem) and whose construction is based on permutation polynomials. One of the reasons of taking this cryptosystem into consideration is the fact, that the MQ-problem is an NP-complete problem. It seems that quantum computers do not have an advantage on solving NP-complete problems—that is an important property, since NP problems such as integer factorization and discrete logarithm—i.e., the problems which are used widely today—are weak against quantum computers due to the Shor's algorithms which run in polynomial time.

Firstly, we start this section by introducing basic terminology and definitions, which have been borrowed from [5].

Let p be a prime, n be a positive integer and $\mathbb{F}_q$ be a finite field of $q = p^n$ elements. A polynomial $f(x)$ in $\mathbb{F}_q[x]$ is said to be *permutation polynomial* if it is a bijection of $\mathbb{F}_q$ onto $\mathbb{F}_q$. In other words, it is a permutation polynomial if following properties hold:

- The function f is onto.
- The function f is one-to-one.
- $f(x) = a$ has a unique solution in $\mathbb{F}_q$ for each $a \in \mathbb{F}_q$.

Let $B = \{\vartheta_0, \vartheta_1, \ldots, \vartheta_{n-1}\}$ be a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. Each element $x \in \mathbb{F}_{2^n}$ can be uniquely expressed as $x = \sum_{i=0}^{n-1} x_i \vartheta_i$, where $x_i \in \mathbb{F}_2$. This way, we can identify $\mathbb{F}_{2^n}$ by $\mathbb{F}_2^n$ and the element $x \in \mathbb{F}_{2^n}$ by the n-tuple $(x_0, x_1, \ldots, x_{n-1})$ where $x_i \in \mathbb{F}_2$. The weight of $x = (x_0, x_1, \ldots, x_{n-1})$ denoted by $\omega(x)$ is defined to be the number of ones in x.

Definition 3. Let $\mathbb{F}_{2^n}$ be a finite field. Let $B = (\vartheta_0, \vartheta_1, \ldots, \vartheta_{n-1})$ be a basis of this field over $\mathbb{F}_2$. Let $\alpha \in \mathbb{F}_{2^n}$ with the corresponding n-tuple $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$. We then define a linearized polynomial (or p-polynomial) $L_\alpha(x)$ on $\mathbb{F}_{2^n}$ corresponding to an element $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ as

$$L_\alpha(x) = \sum_{i=0}^{n-1} \alpha_i x^{2^i}. \tag{2}$$

Also, $Tr(x)$ denotes the *trace* function from the field $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$, i.e.,

$$Tr(x) = x + x^2 + \cdots + x^{2^{n-1}}.$$

As we have already mentioned, the construction of Poly-Dragon relies on permutation polynomials. We state two theorems, which were proved in [5], now.

Theorem 1. *Let n be an odd positive integer, and $\beta = (\beta_0, \beta_1, \ldots, \beta_{n-1})$ be an element of $\mathbb{F}_{2^n}$ such that $\omega(\beta)$ is even and that 0 and 1 are the only roots of $L_\beta(x)$ in $\mathbb{F}_{2^n}$. Suppose k_1 and k_2 are nonnegative integers such that $GCD(2^{k_1} + 2^{k_2}, 2^n - 1) = 1$. Let ℓ be any positive integer with $\left(2^{k_1} + 2^{k_2}\right) \cdot \ell \equiv 1 \pmod{2^n - 1}$ and γ be an element of $\mathbb{F}_{2^n}$ with $Tr(\gamma) = 1$. Then*

$$f(x) = \left(L_\beta(x) + \gamma\right)^\ell + Tr(x)$$

is a permutation polynomial of $\mathbb{F}_{2^n}$.

Theorem 2. *Let n be an odd positive integer, and α be an element of $\mathbb{F}_{2^n}$. The polynomial $g(x) = \left(x^{2^{k2^r}} + x^{2^r} + \alpha\right)^l + x$ is a permutation polynomial of F_{2^n} if $Tr(\alpha) = 1$ and $(2^{k2^r} + 2^r) \cdot l \equiv 1 \pmod{2^n - 1}$.*

Since Poly-Dragon is a public-key cryptosystem, we will now describe the public key generation. Both permutation polynomials $f(x)$ and $g(x)$ from both theorems are used in the process.

Care must be taken when choosing values for parameters listed in both theorems. We will follow the authors' suggestion in [5] for the choice of parameters, which is as follows. We set $r = 0, n = 2m - 1, k = m, k_2 = m, k_1 = 0$, $l = 2^m - 1$ and $\ell = 2^m - 1$. So in this case we will take permutation polynomials

$$f(x) = \left(L_\beta(x) + \gamma\right)^{2^m - 1} + Tr(x) \quad \text{and} \quad g(x) = (x^{2^m} + x + \alpha)^{2^m - 1} + x$$

for the public key generation. Parameters α, β, γ are secret parameters. Suppose now that s and t are two invertible affine transformations. The relation between plaintext and ciphertext is $g\big(s(x)\big) = f\big(t(y)\big)$, where variable x denotes the plaintext and variable y denotes the ciphertext. Suppose that $s(x) = u$ and $t(y) = v$. Thus we have the following relationship between plaintext and ciphertext

$$\left(u^{2^m} + u + \alpha\right)^{2^m - 1} + u = \left(L_\beta(v) + \gamma\right)^{2^m - 1} + Tr(v)\,.$$

Since $u^{2^m} + u + \alpha$ and $L_\beta(v) + \gamma$ are nonzero in the field $\mathbb{F}_{2^n}$, this relation gives

$$\begin{aligned}\left(u^{2^m} + u + \alpha\right)^{2^m} \left(L_\beta(v) + \gamma\right) + u\left(u^{2^m} + u + \alpha\right)\left(L_\beta(v) + \gamma\right) + \\ \left(u^{2^m} + u + \alpha\right)\left(L_\beta(v) + \gamma\right)^{2^m} + Tr(v)\left(u^{2^m} + u + \alpha\right)\left(L_\beta(v) + \gamma\right) = 0\,. \quad (3)\end{aligned}$$

Suppose $Tr(v) = \zeta_y \in \{0, 1\}$. We can identify the field $\mathbb{F}_{2^n}$ with the field $\mathbb{F}_2^n$ (i.e., the set of all n-tuples over $\mathbb{F}_2$ by using a fixed basis $B = \{\vartheta_1, \vartheta_2, \ldots, \vartheta_n\}$ of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. Substituting $u = s(x)$ and $v = t(y)$, where $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$, we get n non-linear polynomial equations which form the *public key* of the cryptosystem.

These equations are in general:

$$\sum a_{ijk}x_ix_jy_k + \sum b_{ij}x_ix_j + \sum (c_{ij} + \zeta_y)x_iy_j + \sum (d_k + \zeta_y)y_k + \sum (e_k + \zeta_y)x_k + f_l = 0\,, \quad (4)$$

where $a_{ijk}, b_{ij}, c_k, d_k, e_k \in \mathbb{F}_2$. They are of degree three and therefore each of them contains $\mathcal{O}(n^3)$ terms. Since we have n equations, total size of public key is $\mathcal{O}(n^4)$. It is further possible to reduce the complexity to $\mathcal{O}(n^3)$ without changing the security of the cryptosystem in polynomial time. What is important to realize is the fact that while the public key is linear in ciphertext variables (y_i), it is non-linear in plaintext variables (x_i).

As we have mentioned, the system of equations (4) forms the public key. The *private key* of the cryptosystem consists of parameters α, β, γ and of two affine transformations (s, t).

Now we will describe the encryption and decryption algorithm as it was proposed in [5].

Algorithm 2: Poly-Dragon: Encryption Algorithm

If Bob wants to send a message $x = (x_1, x_2, \ldots, x_n)$ to Alice, he does the following:

1. Bob substitutes the plaintext variables $(x_1, x_2, \ldots, x_n)$ and sets $\zeta_y = 0$ in public key and gets n linear equations in ciphertext variables $(y_1, y_2, \ldots, y_n)$ over a finite field $\mathbb{Z}_2$. Bob solves these equations (e.g., by Gaussian elimination) and obtains a vector $y' = (y'_1, y'_2, \ldots, y'_n)$.
2. Bob substitutes the plaintext variables $(x_1, x_2, \ldots, x_n)$ and sets $\zeta_y = 1$ in public key and gets n linear equations in ciphertext variables $(y_1, y_2, \ldots, y_n)$ over a finite field $\mathbb{Z}_2$. Bob solves these equations (e.g., by Gaussian elimination) and obtains a vector $y'' = (y''_1, y''_2, \ldots, y''_n)$.
3. The ordered pair (y', y'') is the resulting ciphertext.

Algorithm 3: Poly-Dragon: Decryption Algorithm

Input : Ciphertext (y', y'') and secret parameters $\alpha, \beta, \gamma, s, t$
Output: Message $(x_1, x_2, \ldots, x_n)$

1. $v_1 \leftarrow t(y')$ and $v_2 \leftarrow t(y'')$.
2. $z_1 \leftarrow L_\beta(v_1) + \gamma$ and $z_2 \leftarrow L_\beta(v_2) + \gamma$.
3. $z'_3 \leftarrow z_1^{2^m-1}$ and $z'_4 \leftarrow z_2^{2^m-1}$.
4. $z_3 \leftarrow z'_3 + Tr(v_1)$ and $z_4 \leftarrow z'_4 + Tr(v_2)$.
5. $z_5 \leftarrow z_3^{2^m} + z_3 + \alpha + 1$ and $z_6 \leftarrow z_4^{2^m} + z_4 + \alpha + 1$.
6. $z_7 \leftarrow z_5^{2^m-1}$ and $z_8 \leftarrow z_6^{2^m-1}$.
7. $X_1 \leftarrow s^{-1}(z_3 + 1)$, $X_2 \leftarrow s^{-1}(z_4 + 1)$, $X_3 \leftarrow s^{-1}(z_3 + z_7 + 1)$, $X_4 \leftarrow s^{-1}(z_4 + z_8 + 1)$.
8. Return (X_1, X_2, X_3, X_4).

The decryption algorithm returns following four messages (X_1, X_2, X_3, X_4). One of them is the correct message and it is not difficult to find out, which one.

4. Our construction and results

Finally, we present our construction of the generator MSTg which uses Poly-Dragon to generate the random covers. We wanted to study this construction and to see whether the number of random covers or their structure play an important role in the statistical properties of the generator. In the testing, we have decided to choose similar approach as the authors of MSTg. That is why the size of our generator is practically the same as theirs. Due to the fact that Poly-Dragon requires the number of elements in a certain field to be an odd power of number two, we have chosen the sizes of two groups to be $|\mathcal{G}_1| = 2^{257}$ and $|\mathcal{G}_2| = 2^{129}$.

At first, we generated the random covers. We used the Poly-Dragon cryptosystem in a counter mode: in the beginning we generated a random seed (an element from the finite field) and a random constant which we iteratively added to the seed to create a new plaintext. We encrypted the new plaintext and the resulting ciphertext was the new element of the random cover of the corresponding finite group. Since there is not an available implementation of Poly-Dragon, we have programmed our own. During the process, we have discovered several interesting facts about the cryptosystem itself and have modified the encryption algorithm slightly. For further information see [1].

Our parameters of Poly-Dragon were as follows:

- Irreducible polynomial – randomly chosen of a corresponding degree.
- α, γ – randomly chosen ($Tr(\alpha) = Tr(\gamma) = 1$).
- β – set as $1 + \vartheta$ (in compliance with [5]).
- s, t – both affine transformations were set as identities (for easier implementation).

Using this construction of Poly-Dragon we have generated enough elements for each random cover. It should be noted that the process of generating a random cover is not just a generation of subsets of a group. The collection of subsets $\mathcal{A}$ has to be a cover of the group, which means, that all elements of the group must have a factorization into the elements of the cover. However, this process can be done efficiently and is described in the article [6].

Once we generated random covers for the MSTg generator we simply generated a sufficient number of bits and tested its statistical properties using the NIST Statistical Test Suite [4].

We have tested four constructions of the generator which differ in the number of random covers and in the number of elements in one subset:

- MSTg/Poly-Dragon with 3 random covers, $|\mathcal{A}_i| = 3$, $C = 1$, ver. $\mathcal{A}$,
- MSTg/Poly-Dragon with 4 random covers, $|\mathcal{A}_i| = 3$, $C = 1$, ver. $\mathcal{A}$,
- MSTg/Poly-Dragon with 5 random covers, $|\mathcal{A}_i| = 3$, $C = 1$, ver. $\mathcal{A}$,
- MSTg/Poly-Dragon with 5 random covers, $|\mathcal{A}_i| = 2^8$, $C = 1$, ver. $\mathcal{A}$.

For each construction we generated 100 different instances of a generator––with different random covers of a given type and we used each instance to generate 100 bit sequences of length 10^7 bits. For the statistical evaluation of the sequences we used an implementation of the NIST test suite which has been programmed at our institute at the university. The testing of one sequence consisted of 200 smaller tests which yielded 200 p-values. It is important to say that we tested the generator at the significance level equal to 0.01. We denote one set of 200 p-values as a "p-value set". We have therefore for each instance of a generator 10^9 output bits and one p-value set. We studied the number of "defect" p-values in the set, i.e., the number of p-values that were smaller than our significance level.

Finally, we present the results of the testing.

TABLE 1. NIST test results—MSTg with Poly-Dragon.

Version [bits]	Output [bits]	$\#c$	$\lvert\mathcal{A}_i\rvert$	f_0	f_1	f_2	f_3	f_4	f_{5+}	f_{max}
257/129	129	3	3	38	38	16	6	1	1	8
257/129	129	4	3	42	37	14	2	2	3	7
257/129	129	5	3	41	33	18	5	1	2	6
257/129	129	5	256	47	34	12	4	1	2	6

Each row of the Table 1 represents a certain type of a construction of a generator. Symbol "$\#c$" denotes the number of random covers, $|\mathcal{A}_i|$ denotes the size of one subset of the group, f_i denotes the number of p-value sets that have exactly i "defect" p-values (5+ means five and more). The column f_{max} shows what the maximum number of "defect" p-values was. For example, we can see that in the first construction with 3 random covers, where one subset in the cover had 3 elements, 38 instances of the generator were NOT rejected as a good random number generator and that the maximum number of unacceptable p-values was eight.

It stems from the table that in our case it was more or less not important whether a generator had three, four or five covers. They were evaluated as "good" random number generators with the probability of approximately 40 %.

However, we can see that when we changed the number of elements of one subset in the cover (the 3rd and 4th row of the table), the number of instances which were NOT rejected has risen to 47.

We have carried out one more test—we tested the output of Poly-Dragon to see whether the resulting random cover is truly "random". We generated 100 different instances of Poly-Dragon and generated output sequences of the length 10^7 bits. The results are presented in the Table 2.

Table 2. NIST test results—Poly-Dragon.

Version [bits]	f_0	f_1	f_2	f_3	f_4	f_{5+}	f_{max}
257	20	27	30	13	14	6	8

This is an interesting result, because only one fifth of the instances were NOT rejected by the testing suite. This is a relatively small number when compared to commonly-used PRNGs. However, it is interesting that even though Poly-Dragon itself has a success rate of only 20 %, the MSTg generator which uses Poly-Dragon to generate its random covers has a success rate of 47 %. For comparision, the success rate of the MSTg generator presented by its authors had a success rate from 48 % to 51 %.

5. A few notes on security

In this section we state a few notes about the security of this generator. Since we have not changed the generating algorithm in general, we expect it to have the same level of security as the original generator proposed in [3]. Our change is in the generation of random covers—we have used the output of the Poly-Dragon cryptosystem instead of the output of some random number generator.

The security of the original generator is discussed in [3]. The authors argue that the brute-force attack on the generator would have a complexity of $\mathcal{O}(2^{e_1-e_2-\delta-1})$, where e_1 is the bit-length of elements from the first group $\mathbb{Z}_{2^{e_1}}$, e_2 is the bit-length of elements from the second group $\mathbb{Z}_{2^{e_2}}$ and $0 \leq \delta \leq 2$, thus if e_1 and e_2 are sufficiently large, for example $e_1 - e_2 \geq 100$, then it is computationally infeasible to determine the initial seed of the generator. Another interesting property of MSTg generator is that for two inputs x and x' of the mapping $\breve{\alpha}$ the outputs $\breve{\alpha}(x)$ and $\breve{\alpha}(x')$ differ in approximately half of the output bits. However, an accurate estimate of the complexity of computing the seed s for a given output sequence $(z_1, z_2, \ldots, z_t)$ has still not been made (to the knowledge of the authors) so this might be worth further investigation. The security of the generator itself is based on a discrete logarithm problem.

The elements of the random covers are the outputs of the Poly-Dragon cryptosystem used in CTR mode. We have generated a random seed, encrypted it with Poly-Dragon and the resulting ciphertext was the element of the cover. Then we increased the value of the seed by a fixed constant, encrypted the incremented seed and the result was the second element of the cover, etc. Therefore if an attacker is able to decrypt the element of the cover (or two consecutive elements) then he is able to predict future cover elements. However, the security here relies on the security of Poly-Dragon. It is based on the MQ-problem. This problem belongs to the category of NP-complete problems, therefore we assume that with a correct choice of parameters this cryptosystem is secure against the brute-force attack. Additionally, authors argue in [5] that this cipher is also secure against linearization equation attacks, differential cryptanalysis, Gröbner basis attacks and XL and FXL algorithms.

We argue that the security of the generator is comparable to the security of the original MSTg generator since the changes we have made to the algorithm itself are only minor. The security of the generation of the cover elements relies on the inability of the attacker to break an instance of the Poly-Dragon cryptosystem in real time (thus he is not able to predict/calculate other cover elements). Therefore we believe that if an attack is found that will be able to predict the other cover elements, than it can be used to break the Poly-Dragon cryptosystem. And if an attack is found that will be able to determine the seed s of the generator from the output $(z_1, z_2, \ldots, z_t)$, then, with high probability, this attack can be applied to the original generator MSTg.

6. Conclusion

We have presented our version of the MSTg generator which uses Poly-Dragon cryptosystem to generate its random covers. We have carried out statistical tests and have found out that the structure of a cover plays a greater role in the statistical properties of the output than the number of random covers. This is not surprising, since the authors of MSTg have made a similar statement in their paper. What is surprising is that the MSTg achieved relatively good results even though Poly-Dragon itself achieved worse results. There is still a lot of work to be done. For example one could wonder how would Poly-Dragon behave if the affine transformations were not set to identities, but were also randomly generated. Another interesting research topic might be the further study of the structure of random covers and whether a different choice of parameters of the generator (different counter value, different construction of core function) would yield better or worse results.

Acknowledgement. The authors are grateful to anonymous reviewers for their helpful comments and remarks that helped to improve the quality of this paper.

REFERENCES

[1] HROMADA, V.—VOJVODA, M.: *A note on Poly-Dragon cryptosystem,* in: 14th Conference of Doctoral Students—ELITECH '12, Bratislava, 2012.

[2] MAGLIVERAS, S. S.—STINSON, D. R.—VAN TRUNG, T.: *New approaches to designing public key cryptosystems using one-way functions and trapdoors in finite groups,* J. Cryptology **15** (2002), 285–297.

[3] MARQUARDT, P.—SVABA, P.—VAN TRUNG, T.: *Pseudorandom number generators based on random covers for finite groups,* Des. Codes Cryptogr. **64** (2012), 209–220.

[4] RUKHIN, A.—SOTO, J.—NECHVATAL, J.—SMID, M.—BARKER, E.—LEIGH, S.–LEVENSON, M.—VANGEL, M.—-BANKS, D.—HECKERT, A.—DRAY, J.—VO, S.: *Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* in: NIST Special Publication 800-22, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2010, `http://csrc.nist.gov/rng.`

[5] SINGH, R. P.—SAIKIA, A.—SARMA, B. K.: *Poly-Dragon: an efficient multivariate public key cryptosystem,* J. Math. Cryptol. **4** (2010), 365–373.

[6] SVABA, P.—VAN TRUNG, T.: *On generation of random covers for finite groups.* Tatra Mt. Math. Publ. **37** (2007), 105–112.

Received August 15, 2012

Institute of Computer Science and Mathematics
Slovak University of Technology
Ilkovičova 3
SK–812-19 Bratislava
SLOVAKIA

E-mail: viliam.hromada@stuba.sk
milan.vojvoda@stuba.sk